In-place reconfiguration of lattice-based modular robots

Joel Moreno Vàzquez

Bachelor's Thesis
Director: Vera Sacristán Adinolfi
Department: Mathematics

Computer Engineering Degree
Specialization: Computer Science
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

4th of July, 2019

Abstract

In the 1970s the concept of modular robots appeared in the field of robotics. This consists of robot systems formed by several connected modules which can have different functionalities. These robots can also change their shape by relocating their modules.

Modular robots can be of many types: lattice based, chain based or a mix of both. In addition their modules can be of many different shapes. When relocating, different modules use different movements: compressing, pivoting or sliding. Due to these many differences and the ability to change their shape, modular robots have a large range of possibilities and critical abilities such as adaptability or the ability to undertake many different jobs.

Reshaping is one of the most important features of modular robots but it also is were most of the computing issues arise. To reshape, modular robots can apply many different approaches: self-reconfigurating, self-assembling or disassembling, grasping, enveloping, etc. In this project we focus on self-reconfiguration.

The goal of this project is to design and develop a centralized reconfiguration algorithm for square-shaped lattice-based modular robots whose modules move by sliding, which reconfigures a certain configuration C into C' with the same amount of modules. This reconfiguration process can be divided in three different parts. First of all, reconfiguring C into R_c, which is the resulting configuration from flooding the bounding box of C from bottom to top and from left to right with the same number of modules. After that, we need to reconfigure R_c into $R_{c'}$, the resulting configuration from flooding the bounding box of C'. Lastly, we have to reconfigure $R_{c'}$ into C'. Since the second step is a trivial reconfiguration problem and the third step is the inversion of the first one, in this project we focused only in the first step.

In this project the cost of the algorithm has been analyzed in both a theoretical and a practical way. If n is the size of the input grid and m is the number of modules of the robot, the computation cost of the algorithm is $O(n)$ and the total number of moves of the modules of the robot is $O(m^2)$.

This bound is tight.

In the practical experiments we ran the algorithm with many inputs of different sizes and with distinct structures. The results showed that the closer m was to n, the larger was the execution time. However, when nesting all the movable modules in the depths of the pseudo-holes' hierarchies the greater was the number of steps the reconfiguration had to make.

Resum

Durant els anys 70, va aparèixer en el camp de la robòtica el concepte de robots modulars. Aquest, consisteix en sistemes robòtics que estan formats per diversos mòduls connectats, que poden tenir funcionalitats diferents. Aquests robots, també poden canviar la seva forma, recol·locant els seus mòduls.

Els robots modulars poden ser de molts tipus: arquitectura reticular, arquitectura en cadena o una barreja d'ambdues. A més a més, els mòduls poden variar en la seva forma. Quan es recol·loquen, cada mòdul pot moure's d'una forma diferent: comprimint-se, pivotant o lliscant. És per aquestes diferenciacions i l'habilitat de canviar de forma, que els robots modulars tenen un ampli ventall de possibilitats junt amb habilitats crítiques com l'adaptabilitat o la capacitat de fer feines diferents.

Canviar de forma es una de les característiques més importants dels robots modulars, però també és on la major part dels problemes de computació resideixen. Per canviar de forma, els robots modulars poden fer ús de diferents mètodes: auto-reconfiguració, auto-ensamblament o desensamblament, embolcallament, etc. En aquest projecte ens centrem en l'auto-reconfiguració.

L'objectiu d'aquest projecte és dissenyar i desenvolupar un algorisme de reconfiguració, centralitzat, per a mòduls amb forma quadrada i arquitectura reticular, que es mouen lliscant els uns sobre els altres, el qual reconfiguri una certa configuració C en una altra configuració, C', que tingui el mateix nombre de mòduls. Aquest procés de reconfiguració es pot dividir en tres parts diferents. Primer de tot, reconfigurem C en R_c, la qual és la configuració resultant de inundar la capsa contenidora de C de baix a dalt i d'esquerra a dreta amb el mateix nombre de mòduls. Després d'això, necessitem reconfigurar R_c en $R_{c'}$, la configuració resultant de inundar la capsa contenidora de C'. Finalment, hem de reconfigurar $R_{c'}$ en C'. Com el segon pas es un problema trivial de reconfiguració i el tercer pas és la inversió del primer, en aquest projecte ens hem centrat només en el primer pas.

En aquest projecte hem analitzat els costos del nostre algorisme de forma tant teòrica com pràctica. Si n és el tamany de la graella d'entrada i m és el nombre mòduls del robot, el cost en computació de l'algorisme es $O(n)$ i el nombre total de moviments fets pels mòduls del robot és $O(m^2)$. Aquest límit està acotat.

En els experiments pràctics, vam executar l'algorisme amb diverses entrades de diferents tamanys i estructura. Els resultats mostren que contra més s'acosta m a n, més gran és el temps d'execució. De totes formes, quan tanquem tots els mòduls movibles a l'interior de les jerarquies de pseudoforats augmenta el nombre de moviments realitzats durant la reconfiguració.

Resumen

Durante los años 70, apareció en el campo de la robòtica el concepto de robots modulares. Este, consiste en sistemas robóticos que están formados por diversos módulos conectados, que pueden tener funcionalidades distintas. Estos robots, también pueden cambiar su forma, recolocando sus módulos.

Los robots modulares pueden ser de muchos tipos: arquitectura reticular, arquitectura en cadena o una mezcla de ambas. Además, los módulos pueden variar en forma. Cuando se recolocan, cada módulo puede moverse de una forma distinta: comprimiendose, pivotando o deslizándose. Es por estas diferenciaciones y la habilidad de cambiar de forma, que los robots modulares tienen una extensa variedad de posibilidades junto con habilidades críticas com la adaptabilidad o la capacidad de realizar trabajos distintos.

Cambiar de forma es una de las características más importantes de los robots modulares, pero también es donde residen la mayor parte de los problemas de computación. Para cambiar de forma, los robots modulares pueden hacer uso de distintos métodos: auto-reconfiguración, auto-ensamblaje o desensamblaje, envolvimiento, etc. En este proyecto nos centramos en la auto-reconfiguración.

El objetivo de este proyecto es diseñar y desarrollar un algoritmo de reconfiguración, centralizado, para módulos con forma cuadrada y arquitectura reticular los cuales se mueven deslizándose unos sobre otros, el cual reconfigure una cierta configuración C en otra configuración, C', que tenga el mismo número de módulos. Este proceso de reconfiguración se puede dividir en tres partes diferentes. Primero de todo, reconfiguramos C en R_c, la cual es la configuración resultante de inundar la caja contenedora de C de abajo a arriba y de izquierda a derecha con el mismo número de módulos. Después de esto, necesitamos reconfigurar R_c en $R_{c'}$, la configuración resultante de inundar la caja contenedora de C'. Finalmente, reconfiguramos $R_{c'}$ en C'. Como el segundo paso es un problema trivial de reconfiguración y el tercer paso es la inversión del primero, en este proyecto nos hemos centrado solo en el primer paso.

En este proyecto hemos analizado los costes de nuestro algorismo de manera tanto teórica como práctica. Si n es el tamaño del tablero de entrada y m es el número de módulos del robot, el coste en computación del algoritmo es $O(n)$ y el número total de movimientos hechos por los módulos del robot es $O(m^2)$. Este límite está acotado.

En los experimentos prácticos, ejecutamos el algoritmo con distintas entradas de diferente tamaño y estructura. Los resultados muestran que contra más se acerca m a n, mayor es el tiempo de ejecución. De todas formas, cuando encerramos todos los módulos móviles en el interior de jerarquías de pseudo-agujeros aumenta el número de movimientos realizados durante la reconfiguración.

Contents

Introduction

Robotics is one of the most extensive research areas in the engineering field and throughout the years has given birth to many different kinds of robots such as: static, kinematic, autonomous, pre-programmed, etc; and with many different purposes: domestic chores, building, transportation, entertainment, etc. All these robots have many characteristics that make them suitable for their different tasks, but, due to their typically fixed-morphology, the different jobs that a robot can execute are defined by its size, shape and other physical attributes that can become restrictions at some point. In order to overcome these restrictions, self-reconfigurable robots have been developed.

When we talk about modular robots, we refer to robots that are built of several modules. These can be identical or different and are usually interchangeable due to their identical connections. Each module composing a modular robot can have many different purposes. This together with being able to reconfigure and change the structure, makes a single modular robot capable of undertaking several different jobs. Modular robots can be of different kinds: chain, lattice or hybrid. Lattice-based modular robots can be also classified by their many distinct shapes, however, the most common ones are square and hexagonal. In addition to their shape, during reconfiguration, modules can move in many different ways, the most important ones are: sliding, pivoting and compressing.

As described in [7] there are three main reasons that motivate the study and evolution of this branch in robotics: the versatility, the robustness and the low cost of modular robots. First of all due to their ability to change their structure and their high mobility, modular robots are capable to adapt themselves in order to undertake different jobs in many different situations. In static built or fixed structure robots, this could be found as a problem due to having to design a different robot for each different situation or job. Second, when we talk about robustness we refer to the ability to swap a malfunctioning module with a working one giving the robot the ability to maintain and fix itself. And last, since a robot can be formed of several identical modules, with the right materials we can reduce their production

cost. These main reasons give the needed motivation to develop and evolve modular robots.

Our project works within this framework. We will use lattice modules that are square-shaped distributed over a grid on the plane. An example of modular robot suitable for our project would be the EM-Cubes from 2008, see Figure 1. This is a modular robot with cubic modules that has the ability to arbitrarily change its structure. Its modules are able to swap thanks to their magnetic connectors which allow them to assemble, disassemble and communicate.

Figure 1: Modular robot EM-Cubes from 2008, extracted from [9].

As for how does the robot reconfigure, the modules move by sliding relative to their neighbouring modules. When closely studying these movements we can identify to different moves: from side to side and around the corner (see Figure 2). The first one consists in sliding over the neighbour modules' surface only vertically or horizontally. The second one consists in combining a vertical and horizontal movement in order to move around the corner of a neighbour module.

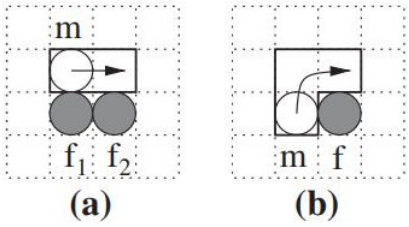

Figure 2: (a) an example of straight sliding, (b) an example of sliding around a corner, extracted from [1].

Previously we described the main characteristics that motivated the development of modular robots, but, despite these great features, there also exist a few issues that are under study in order to solve them and improve

them. The main problems found are: the nonexistence of a general purpose module model that serves as the foundation, improvements in resistance, strength and movement speed, and the limited amount of modules a single robot can control. These can be considered some of the physical or technical issues that are under improvement.

As mentioned in [5], there exist many different processes a modular robot may need to do. Self-reconfiguration, self-assembly, self-adaptation and grasping are just a few of them. Each of these processes requires an algorithm designed with that specific purpose and its approach may arise different problems. For instance, if we are working on a distributed algorithm, we may have to find a solution when two modules are trying to move to the same cell at the same time. Another example could be the inability to reach certain configurations given certain frameworks. If we work on a 2D environment with square modules that move by pivoting, starting with a structure that resembles a Palmier, blocks the mobility of all modules.

In our project, we will be experimenting with self-reconfiguration and some of its issues. The main one we encountered has been dealing with holes. During the reconfiguration process we have to fill cells that are closed within a hole that has no movable modules in it. When this happens we need move modules in other holes that will help us free modules in the desired one.

More precisely, in this project we aim to design and implement an algorithm based on the one in [1] but with a few distinctions. The first one is the goal, we aim to reconfigure our robot into filling from bottom to top the rows delimited by its starting minimum bounding box. This way, instead of turning the figure into a straight line we will melt down the structure flooding the bottom rows. This change of goal allows us to aim for a second objective which is to make the reconfiguration in-place. The aim for that is to use as minimum space as we can, with this purpose in mind we will only use the area delimited by the initial bounding box and an extra module space on each side to move around.

This report is structured as follow. In Chapter 1 we describe the context of this project, its goals and previous studies. Following in Chapter 2 we briefly explain the functioning of the algorithm in [1]. Next in Chapter 3 we describe the algorithm designed in this project and the method created for its visualization. In Chapter 4 we proof the validity of the algorithm. And finally, in Chapter 5 the methodology and planning used in the the project are explained.

Chapter 1

Goal and context

1.1 Context

In this section we locate our problem within the extensive world of robotics, distinguishing the different aspects that are involved in this project, such as modular robots and their different types of reconfiguration algorithms. And we also describe which agents take part in the project.

1.1.1 Research Fields

In the following two sections, we explain which concepts in the field of robotics and algorithm design are involved in this project for the reader to be able to understand the totality of the project.

Modular Robots

For the last two decades modular robots have been a growing area of research, evolving from proof-of-concept systems to physical implementations and simulations. There exists a whole world of different modular robots with different architectures, shapes, moves and other variable features and capabilities. Our area of study consists of modular robots with lattice architecture, square-shaped that move by sliding. Due to the high complexity of reconfiguration in 3D, we restrict the problem to 2D configurations. For a better understanding of the environment of our work, we explain in more detail the features of the robots that are used.

Modular robots as their name implies, are composed of many modules of the same kind or from a small repertoire, which share common docking interfaces that allow them to achieve adaptive configurations. Their connection allows them to transfer also electrical power and communicate. Even though it is also possible for these robots to possess specialized modules such as: wheels, claws, sensors, cameras, etc, in this project we work

with a completely uniform modular system made of robots of the same kind.

The architecture that we focus in is lattice-based, which means that the modules organize themselves in regular shapes such as a square or an hexagonal grid. Our main purpose is designing an algorithm for square-shaped robots which fit on a square grid.

Another feature that is key for the understanding of the problem is the robots movement. A variety of robots have been designed giving them different ways to move and reconfigure. The main ones used are:

- **Compressing**: Consist of flexible modules able to compress and expand themselves, pushing the other modules around, therefore rearranging their configuration.
- **Pivoting**: When using regular shapes, all modules are connected through at least one of the shape's faces. By rotating about any of the connected corners of the module these can relocate themselves.
- **Sliding**: This movement does not modify neither the volume, nor orientation of the modules. These advance by sliding along the perimeter of the configuration.

The different types of movements can be seen in Figure 1.1.

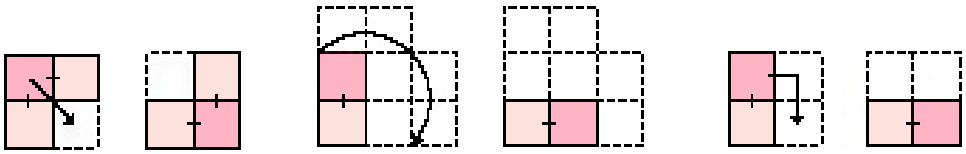

Figure 1.1: Illustration of compressing, pivoting and sliding movements.

We have chosen to work exclusively with sliding modules because the same problem proposed in this project has already been solved on compressing modules and it is known that there are a few configurations that are dead-locks when we refer to pivoting movement since this makes us of more space, than the one represented by the module, when rotating.

Algorithm Approaches

When confronting a reconfiguration algorithm, there are a few characteristics that need to be discussed. The first thing is to decide whether we want serialized moves or parallel moves. If we decide on serialized moves the algorithm, assuming it is well implemented, will for sure take more time since we will only be able to move one module at a time. Despite taking more time,

a sequential algorithm guarantees that there will be no collisions between moving modules along the reconfiguration. On the other hand choosing a parallel movements might allow reaching the final configuration faster, but it can also take longer due to unexpected crossed paths of distinct modules.

The second feature to discuss would be whether we should use a centralised or a distributed algorithm. Using a centralized algorithm means that the computing processes take part in a central controller, and that we will need the information of the whole robot and know from the beginning the complete configuration. This kind of approach allows us to easily find a solution since we know the resulting configuration of each move. A different way of solving the problem could be using a distributed algorithm which allows robots to know about their close surroundings and command their close modules in order to arrive to the desired configuration.

After discussing the pros and cons of each feature, we decided to design a centralized sequential algorithm. This can be the basis for a distributed algorithm later on.

1.1.2 Stakeholders

Designing an algorithm can not begin without people who are interested in the expansion of the research field, expecting to use the algorithm with a defined purpose or willing to work on it.

- **Developer:** The developer is the person who has designed and implemented the algorithm using which means he felt necessary, searching for information on the topic and in the end proving the correctness of the algorithm and the validity of the entire project. In this project the developer was myself.

- **Director:** The director, in this project Vera Sacristán Adinolfi, has been in charge of supervising and advising the developer for him to be able to do an accurate research and find a solution to any possible problem that could appear throughout the development.

- **Benefited Actors:** Since the main objective of this project is to expand the knowledge and tools developed in the field of modular robots, it's beneficiaries will be the researchers interested in this area who will be able to apply the resulting algorithm to their own projects, moreover, they will be able to use found information in this project to develop a more complex algorithm or also continue with the further improvements.

1.2 Scope

The aim of this project is to design and develop an algorithm that is able to rearrange in-place any 2D modular configuration of sliding squares into a simpler and very specific one. By in-place we mean that the robot will never leave the 1-offset of its bounding box. This will be a nice feature of the algorithm, since it will allow the reconfiguration to happen in restrictive space conditions.

The starting configuration can be delimited by a minimum bounding box such that at least one module is in contact with each limits of the box. In our case we will be working with square modules, what reduces the minimum bounding box to a rectangle. Knowing that bounding box, our purpose is to arrange the modules in such a way that all possible rows are filled from the bottom to the top as if we had melted the whole structure until it filled the box in the most uniform way. Since we can not ensure that the number of modules will be a multiple of the bounding box's width, it may happen that the last row is incomplete, in which case we will group the modules in the top row as much to the left as it is possible.

An important aspect to point out is that from the beginning until the end, we will always have only one connect component and that any move that disconnects one or a group of modules from the rest of the robot will be illegal as they must stay connected all times.

As mentioned in Section 1.1.1, our approach to the algorithm will be to design it sequential and centralized, what not only will ensure that we find a way to rearrange the modules, but it will make easier the validation process since we will be able to study all the decisions and moves made step by step.

In order to be able to show and demonstrate how the algorithm works we have also developed a simple web page that given a valid arrangement of the modules, runs the algorithm and shows step by step all the moves of the modules from one configuration to the other.

1.3 State of the Art

In this section we give some more historical details on the state of the art of the distinct components that take part in our project.

1.3.1 Modular robots evolution

The concept of Modular Robots appeared in the 1970s with the "quick change" end effector and automatic tool changers. But, the first robot to be based completely in the common connections mechanism was the CEBOT

developed by Toshio Fukuda in the 1980s [7].

From then on, along the years, modular robots evolved giving birth to the first lattice-based modular systems in the 1990s and another kind of modular robots with a chain-based architecture (see Figure 1.2).

Starting from that, robots' size was reduced, some researches at universities simplified the internal hardware, performance was improved and many prototypes were built in order to prove that modular robots had much more versatility, were more robust and the production cost was lower.

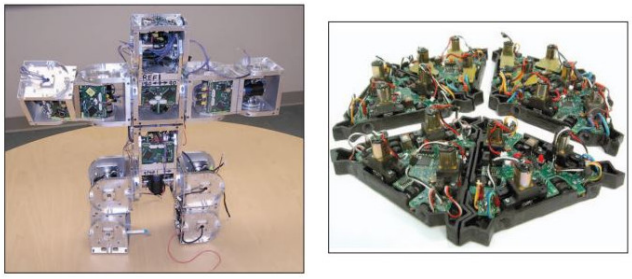

Figure 1.2: On the left a robot created by combining square lattice and chain modules, on the right, triangular modules that rearrange by themselves on an air hockey table, both of them extracted from [7].

That said the future steps in the field are believed to be an increment of the number of modules the system can sustain or control, the development of an optimal and general purpose module design and to expand the physical limits in strength, motor power, energetic efficiency, dexterity and other aspects [7].

1.3.2 Designed Algorithms

As for the algorithms designed, in the 1990s with the apparition of the first lattice-based robots, distributed algorithms were studied in order to be able to manage robots with a huge amount of modules.

The field of modular robots reconfiguration is vast so we will focus on 2D algorithms for lattice-based robots. On that line we can find some recent studies on reconfiguration for both square and hexagonal-shaped since those are the most common regular forms described when talking about lattice architecture modules. As mentioned in the Section 1.1.1 there are various movements that robots can do to move themselves producing a new configuration. Many studies investigate the properties and characteristics

of the different movements. When restricting to sliding modules, one of the most relevant ones is [1], which proves that in a model consisting of square modules which are allowed to slide about one another any two configurations of the same number of modules can be transformed into each other by a sequence of moves. Its 3-dimensional counterpart is [2]. Another recent study relevant to our work is a distributed algorithm of general purpose for both square and hexagonal modules [8]. Although in this paper the modules can squeeze, the proposed algorithm traverses the external boundary of the robot, which is also the appropriate strategy for sliding modules. A different kind of work in lattice robots would be [4] which does not implement a mobility algorithm but studies a characterization of the reconfiguration space of self-reconfiguring robotic systems for hexagonal modules that may help improve planning efficiency in mobility algorithms.

Future advances on the field are expected to be algorithms for parallel motion in large-scale robot systems, for optimal reconfiguration in time or energy, improvement on the efficiency and scalability, etc.

Chapter 2

Pushing Squares Around

Pushing Squares Around (PSA) algorithm ([1]) is a reconfiguration sequential algorithm for square-shaped modular structures in the plane. This algorithm serves as base for our project due to its similarities. However, during the design of our algorithm we also noticed some differences. For this reason we briefly describe the PSA algorithm.

First of all, the goal of PSA is to transform any modules connected configuration into a horizontal strip rooted at the rightmost module in the initial configuration. For this, first of all we need to process the input so we can extract any data or representation of the map needed. The first thing to extract will be the interconnection between modules.

There exist several ways to represent in a graph the adjacency in a configuration since we might not need to know all of them. In the PSA algorithm two kinds of graphs are simultaneously used to get all the information from the configuration. The first one, a complete adjacency graph describing all the adjacencies between modules, we will call it G. And the second one, the cactus graph of G. A cactus graph is a tree of maximal cycles, which is rooted at the origin of the horizontal strip (see Figure 2.1 for an example). A cactus graph, always has a leaf, which can be either a single module or a maximal cycle. In case the leaf is a single module, it must be movable. Otherwise, the leaf would be a maximal cycle and therefore must have a movable module.

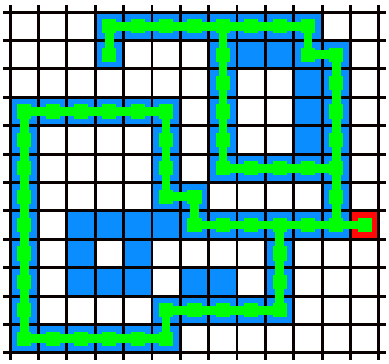

Figure 2.1: Example of CG, in green, for a configuration, extracted from [1]. See its root in red at the origin of the strip.

Any movable module, can advance forever along the boundary belonging to the hole the module is in contact with. Therefore, either the module moves along the outer boundary and can reach the tip of the strip or it can be moved to a position which creates a bigger maximal cycle (see Figure 2.2 for an example). In this last case, the maximal cycle created encloses the one the module belonged to. Therefore, with each new maximal cycle we are getting closer to the outer boundary, whose modules can reach the end of the strip.

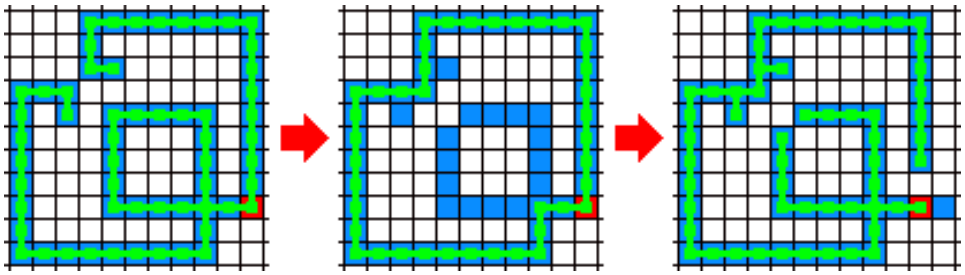

Figure 2.2: Example of complete process to extend the strip rooted at the red module. See in green the cactus graph and its modifications.

Repeating the described process for each cell in the strip from left to right, will lead us towards the algorithm's goal.

Chapter 3

Reconfiguring within the bounding box

In the previous chapter, we explained the algorithm PSA [1], which, given a certain initial configuration, relocates the modules to form a straight line with all the modules. In our algorithm, instead, we want to compact the modules to the bottom of the minimum bounding box that contains our initial configuration.

The reconfiguration has three different phases (see them in Figure 3.1). The first one consists in building up the bottom row of the bounding box. This gives us a solid base from which we are able to build up the rest of the rows. The second phase comprehends all the rows, from bottom to top, that can be filled completely. Since the number of modules may not be a multiple of the base width, it is possible that the top row is not completely filled. Therefore, in the third phase, the top row is filled from left to right until no more modules are in the wrong place.

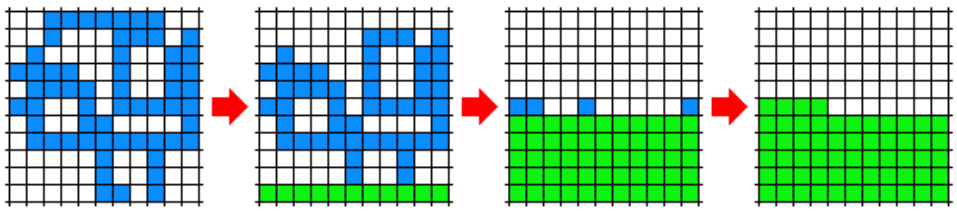

Figure 3.1: Transition stages between the three reconfiguration phases.

Since the second and third phases are based on the methods used in the first one, when explaining the reconfiguration process we focus in this first phase.

As we mentioned previously, the first phase of the reconfiguration consists in filling the bottom row of the bounding box. In order to do that, we choose one movable module and move it around the configuration until it reaches the designed cell. Notice that not all modules can be moved. Since we do not want to split the modules' structure into more than one connected component, we can only move those modules that are not cut vertices of the adjacency graph G of our configuration. We identify such movable modules similarly to PSA. The process to fill a cell, though, is very different as we will see. Recall that all the movements made do not exceed the space needed for a module to move around the bounding box of the configuration.

To completely fill the bottom row, we considered starting to fill the cells from left to right, but that presents an issue. It is possible that the left most cell of the bottom row is not reachable at first, Figure 3.1 is an example of it. However, observe that, because we are using the minimum bounding box, in all the extreme rows and columns, there exists at least one module in the initial stage. This gives us a starting base module from which we can fill the whole row. The described issue is the main difference between phase one, and phase two and three; for, after filling the base of the box, all the cells in the above row, can be reached from at least one module. Therefore, it became the reason why we decided to split the algorithm in three parts: the preprocessing, the filling of the first row, and the filling of the rest of the rows. You can find the algorithm implementation at: **https://github.com/joelProj/Flooding-the-configuration**

3.1 Preprocessing

At the beginning, the algorithm processes the input in order to extract the necessary data. The input received by the algorithm is a matrix representing the cells inside the rectangular bounding box of the initial configuration. Each cell in the matrix is filled with 0 or 1 depending on whether it is empty or it contains a module respectively. This allows us to explore which modules are connected to others, if there are holes in the configuration and with further computations, which modules can move and which can not. The main data structure that we use to store the extracted information is a matrix the same size of the input one, but this time, each cell contains further information about what it contains. More precisely: in case it is an empty cell it contains the number of the hole it belongs to, otherwise, it contains a value to know if the module is movable and which boundaries go across the module and how many times. These boundaries will be explained later with further detail.

A hole can be described as a vertex-connected component of the empty

space. This disconnection causes a hole to be surrounded by modules that conform a cycle in the adjacency graph G of the modules. See Figure 3.2 left for an illustration. In addition to the holes, another kind of empty cells independent set can be found, we call this pseudo-holes. Opposed to the hole definition, a pseudo-hole is a edge-connected component of empty space not surrounded completely by modules.

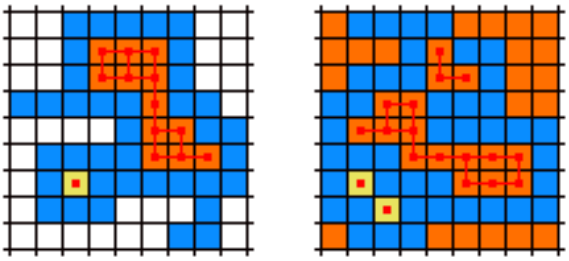

Figure 3.2: Left: a configuration with two holes. Right: a configuration with 2 holes and 5 pseudo-holes (one of them being the unbounded space). All the modules are depicted in blue.

In order to detect the holes and pseudo-holes, first we will surround our input by an extra empty cell. This will connect all the empty cell sets that are in contact with the bounding box edges creating what we call the unbounded empty space. The reason to name it like that is because if we placed our configuration in an unbounded plane, we would observe that this set of empty cells extends to infinity. Then, to identify the different holes we first mark the cells corresponding to the unbounded space by doing a Breadth First Search (BFS) starting at the top left cell of the extended matrix, which we know is an empty cell that belongs to the unbounded space. After that, knowing which empty cells have been traversed and which have not, we will repeat the process from a non-visited empty cell, identifying another hole in the configuration. This process is be repeated until no more empty cells need to be visited. This way we have tagged each empty cell with the label of its respective hole. See Figure 3.3 for an illustration.

Going back to the pseudo-hole definition, it is mentioned that at least one corner of the hole needs to be open so that the hole is not completely surrounded by modules. These openings are called critical pairs following the notation from [1], and their description is: a pair of modules aligned diagonally that are not connected by any other module. See Figure 3.4 for an example.

These critical pairs are important because they allow us to form a hierarchy of holes from outside to the inside using the critical pairs as connections or gates between holes.

To compute this hierarchy, the algorithm first looks through the grid

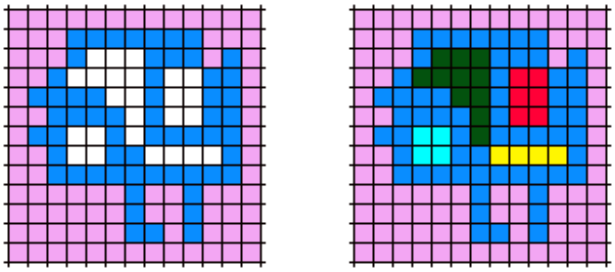

Figure 3.3: A configuration in blue. On the left the unbounded space and on the right all the different holes.

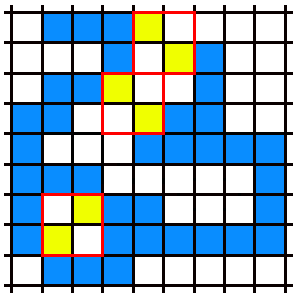

Figure 3.4: A configuration in blue with three critical pairs circled in red and in yellow the modules belonging to them.

listing all the critical pairs that exist and tagging the modules they are formed by. Then starts by looking for critical pairs that are in contact with the unbounded space. Doing this, we get to know which pseudo-holes are children of the unbounded space and mark them with this information. We do the same for each child found. This ends in the construction of the main hierarchy of pseudo-holes in the configuration. Aside from this main hierarchy, there may exist other internal connections between pseudo-holes that are in no way connected to the unbounded space. These are tagged as children of each other when they share a critical pair, grouping them in a same hierarchy level (see holes 4, 5 and 6 in Figure 3.5). However, the process used to find these last connections will be to take a hole that is out of the hierarchy, and execute the same process as for the main hierarchy.

The last step of the preprocessing, consists in finding which modules are movable and which are not. With that purpose, first we find the boundary defined by the modules that are in contact laterally or diagonally with an empty cell, we will call this the traversal boundary. To find this boundary we take one of the modules that are in contact with a hole and then traverse the modules surrounding the hole following the left-hand rule. This implies that the outer boundary is traversed counter-clockwise, while the

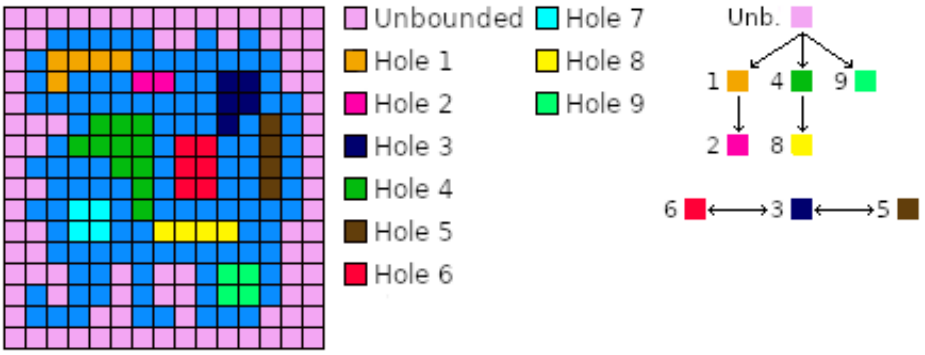

Figure 3.5: Example of hierarchy of pseudo-holes from a configuration.

holes' boundaries are traversed clockwise. This way, we find a different traversal boundary for each hierarchy encountered (see Figure 3.6 and its respective movable modules in Figure 3.7). During this process we count how many times do we go across the same module in a same boundary. This value shows us if a module is movable or not. If the count of a module for a boundary is higher than 1, it means the module is a cut-vertex in the adjacency graph, so, if it is disconnected it will split the structure. Once we know which modules are movable by the traversal boundaries all that remains is reassuring that the movable modules are not physically blocked and can actually slide. This verification is done by checking if the movable modules are completely surrounded by modules.

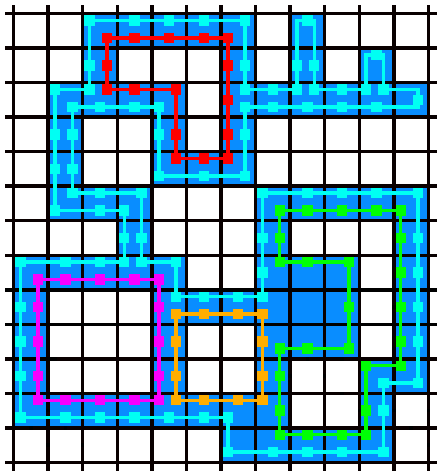

Figure 3.6: Traversal boundaries of a configuration.

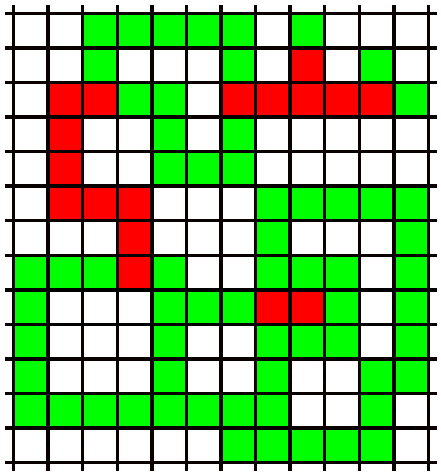

Figure 3.7: Movable modules depicted in green.

After this process, we have gathered all the needed information and can

move on to the reconfiguration of the modules.

3.2 Filling of the first row

The reconfiguration begins filling the bottom row in order to create a solid base from which to start building the rest of the rows. For that we must first find the leftmost module in the bottom row by traversing its cells from left to right. This module will be our starting point. From here we start filling the cells from the module to its left until we reach the bounding box limits. Once we reach the limit, we do the same to the right.

To fill each cell we need to first find a movable module that can reach the designed cell. To do this, we look through the movable modules until we find one in contact with the same hole as the designed cell. If a module meeting the conditions is found the algorithm moves it to the designed cell.

When no module that meets the conditions is found, it means that the cell we want to fill belongs to a pseudo-hole. That is because if the cell belonged to a hole, it would be surrounded by a cycle of modules, therefore, there would always be at least one module in the cycle that is movable without disconnecting the structure. Since we are treating a pseudo-hole, that means it belongs to a certain hierarchy. We call this pseudo-hole the starting hole. Making use of that hierarchy we move through the holes it is connected and the ones connected to these recursively until we find a hole with a movable module. Such module can connect the critical pair we crossed in order to get to that hole. This way, we either generate a cycle containing many pseudo-holes which includes the starting one, or, we reduce the hierarchy and generate some movable modules in a hole closer to the starting one.

Reproducing the described actions in a recursive way we achieve to free modules that are in contact with the starting hole, thus allowing us to fill the desired cell. Figure 3.8 illustrates this process.

Once the cell is filled with a module, the module is tagged as fixed so it can not move anymore. This way we create the solid base of modules in the first row which are unmovable. Having finished the first row we can now move to the last part of the algorithm.

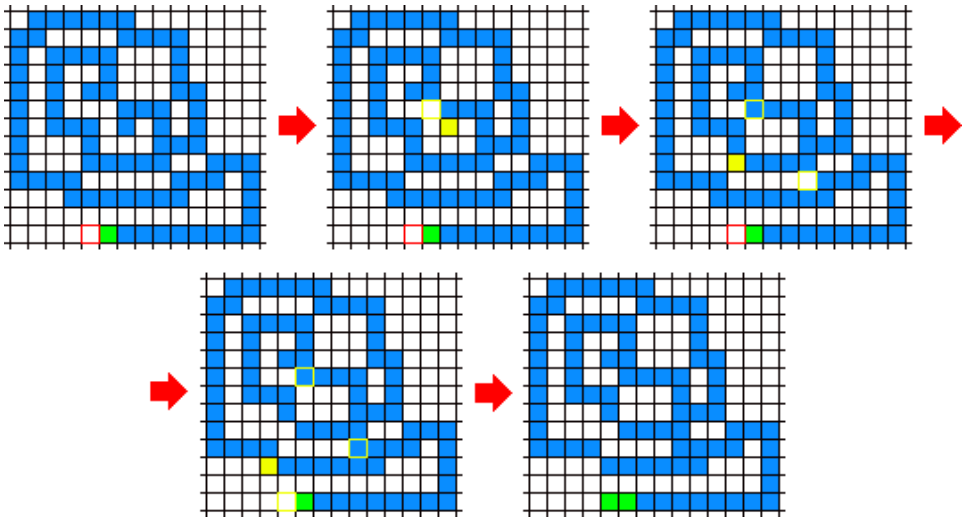

Figure 3.8: Recursive process to send a module to the first row.

3.3 Filling of the rest of rows

In the last stage the first thing to do is to compute how many rows will we be able to fill with the modules on the grid. For that we will use the total amount of modules and the number of cells in a row. These values will give us also the resting number of modules that will belong to the last row.

This section shares the process from the previous one with a few differences. The first one is that the rows can always be filled completely from left to right. That is due to the row below the one we are filling, always being full. This provides a connection with another module at any cell in the row.

Another distinction between both sections is that in the first row, all modules must belong to the outer boundary. Instead, in the rest of the rows it is possible that a cell is located in a hole.

Despite these differences, recursively using the previously described process in each row from left to right leads us to the algorithm's goal. Once all modules are fixed in-place, the reconfiguration will end and we will have reached the last stage.

Chapter 4

Correctness and complexity

4.1 Correctness

Theorem 1. *Let C be any configuration on the plane and R_c a configuration with the same number of modules filling the bounding box of C from bottom to top, left to right. Our algorithm reconfigures C into R_c.*

Remark. We will two lemmas in order to prove Theorem 1.

First of all, we will prove that there always exists a module that is able to slide in the configuration.

Lemma 1. *As long as not all the modules are in their final destination in R_c, there always exists a module that can slide without disconnecting the configuration.*

Proof. In case the configuration contains a module with degree one, that module is movable since it is a leaf in the adjacency graph, therefore its movement does not affect the number of connected components in the configuration. Otherwise, if all modules have a degree of at least two, that implies that there exists a cycle A in the configuration graph G which also is a maximal cycle. Therefore, as proved in [1], there exists a module in the cycle that can slide while the configuration's connectivity is maintained.

In Chapter 3 we mentioned that in some cases no movable module can reach the desired cell. In this cases, a way to liberate modules in the same boundary as the goal cell is needed. The solution we found consists in connecting specific critical pairs with a movable module.

Lemma 2. *If the chosen movable module cannot reach the goal cell, c, it connects a critical pair CP_i that reduces the size of the cycle in contact with*

the hole that contains c.

Proof. When no movable modules are encountered in contact with the desired hole H, it means that none of the modules that belong to the boundary of H are leaves neither belong to a cycle. It also means that H is a pseudo-hole. Otherwise the modules of its boundary would belong to a cycle and therefore due to *Lemma* 1, there would exist a module that could reach c. By systematically undergoing a BFS on the hierarchy were H belongs we find the first critical pair CP_i able to be connected. When CPi is connected we are either creating a cycle or reducing the size of an existing one. In addition, since we are modifying the traversal boundary of H, when we reduce or create a cycle from any pseudo-hole in the hierarchy we are affecting the modules in the boundary of H. Therefore, reproducing the process recursively ensures the liberation of modules in the boundary of H which are able to reach c.

4.2 Complexity

In this section we discuss the computational cost and the number of steps that the whole reconfiguration takes. A step is the action to slide a module to an adjacent free cell. Let n be the number of cells in the input grid, and let m be the number of modules in the input robot. Notice that $m \leq n$.

4.2.1 Computational cost

Preprocessing. In the preprocessing the different steps described and their different costs are as follows:

- Create the configuration matrix $O(n)$.
- Identify holes $O(n - m)$.
- Identify critical pairs and tag hierarchies $O(m)$.
- Identify traversal boundaries $O(m)$.
- Identify movable modules $O(m)$.

Adding up all the costs in the preprocessing we determine that its computational cost is $O(n)$.

Reconfiguration Since the filling of the first row and the filling of the rest of rows use the same process we analyze them together. The following steps are repeated for each module. Hence, the final cost must be multiplied by m.

- Find movable module $O(m)$.
- Move module to desired cell $O(1)$.
- Update configuration matrix $O(1)$.

Therefore, the reconfiguration cost is $O(m)$.

Thus, the computational cost of the algorithm is $O(n + m) = O(n)$.

4.2.2 Number of steps

The number of steps needed for the reconfiguration is the most important measure for a reconfiguration algorithm. Since the movement of each module is time consuming, the number of steps is actually more relevant than the computation cost discussed in the previous section.

Our algorithm uses $O(m)$ steps per module, since each module may have to slide over each other module in order to reach its goal destination. This is evident in the cases when the moving module reaches its destination without having to stop at a critical pair. When it does stop, its moves can be changed to the module that will finally reach the goal position. In other words, achieving that a module fills an empty goal position requires $O(m)$ steps. Therefore, the entire reconfiguration takes $O(m^2)$ steps. This is optimal, as proves the case depicted in Figure 4.1, left, where the robot is made of two strips, each of length $m/2$. Notice that in this case the distance between the top horizontal row and its final destination (one of the two bottom rows) is equal to $m/2$. Therefore, the number of slide moves required to reconfigure is $\Omega(m)$ per module, i.e. $\Omega(m^2)$ overall.

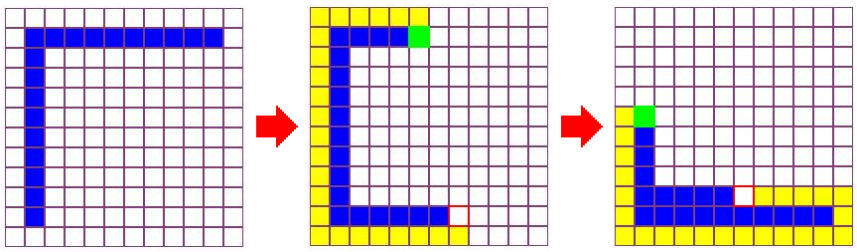

Figure 4.1: Example of reconfiguration requiring $\Omega(m^2)$ slide moves.

Chapter 5

Simulation

5.1 Algorithm Visualization

It is for the purpose of visualization that we have also put up an on-line simulator showing all the movements that the robot would be undertaking step by step from the start configuration to the goal. The simulator allows any user to draw an initial configuration in a grid and then see the steps the robot does to reach the final stage. The simulator can be found at: **https://dccg.upc.edu/people/vera/teaching/tfm-tfg/flooding/**

First of all, the user will encounter a predefined grid followed by two editable values, the number of rows and the number of columns. By pressing the button labeled "Set Values" the grid will resize using the new values introduced by the user (see Figure 5.1 for an example). The size defined for the grid will be the size of the bounding box of the robot too.

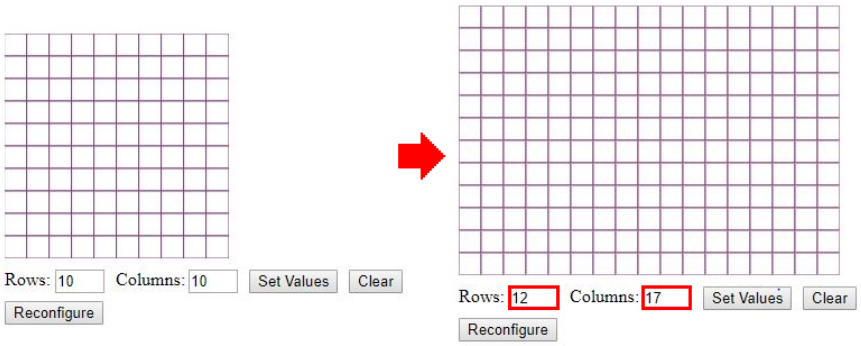

Figure 5.1: Example of grid resizing.

Once we have decided the size of the bounding box we can proceed to draw the initial configuration. By clicking on each cell in the grid we can

turn it blue indicating that the cell contains a module. Clicking again on the same cell turns it white again indicating that it is empty. In case we want to clear the grid and turn all cells empty again we can click the button "Clear". After drawing the initial configuration we can click on the button labeled "Reconfigure". If the robot represented is not fully connected or does not reach all the bounding box limits, a red message appears to warn the user, as shown in Figure 5.2.

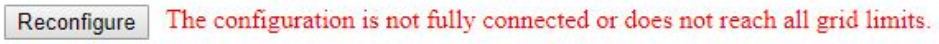

Figure 5.2: Simulator warning message.

Otherwise, the initial stage is turned into a valid input and sent to the algorithm. Once the algorithm has ended, the web page changes into the display view in order to show the steps of the reconfiguration. In the display view we will find the designed initial configuration surrounded by an extra cell to allow the movement of the modules. To move along the reconfiguration steps we need to press the buttons "Prev" and "Next". Between both buttons we find the number of the current step of the algorithm and the total amount of steps. Each module movement is shown by highlighting its moving module in green, the path it follows in yellow and the cell to fill circled in green. See Figure 5.3 for an example of different steps.

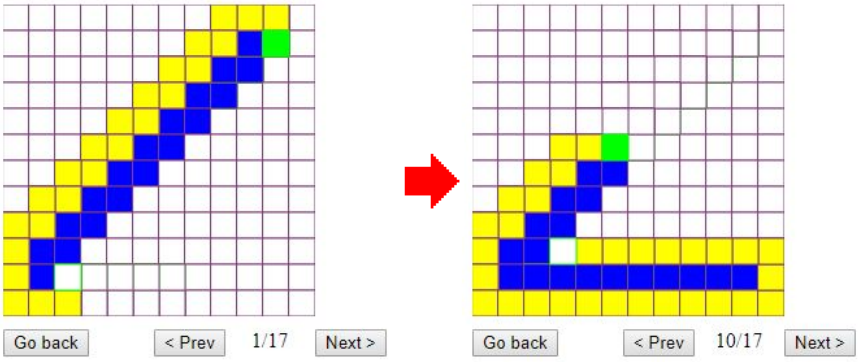

Figure 5.3: Two different steps from a reconfiguration.

Finally if we want to try another configuration we can press the button labeled "Go back" to return to the edit view with an empty grid.

5.2 Experimental Results

In addition to the formal proofs, we have designed several test input configurations to validate in practice the performance of our algorithm. The input tests have been designed with the acquired knowledge during the project.

The experiments consist in running three times the algorithm with each input test and for each test to compute the mean computing time and the number of steps in the reconfiguration. To study the different aspects of a reconfiguration we designed six different types of input tests as follows:

- **Dense:** Consists of configurations that have a great number of modules, close to the number of cells in the grid. See Figure 5.4 for an example.

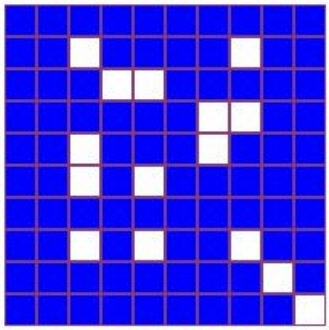

Figure 5.4: "Dense" configuration with size 100.

- **Hierarchy:** Consists of configurations that contain several or large hierarchies of pseudo-holes. See Figure 5.5 for an example.

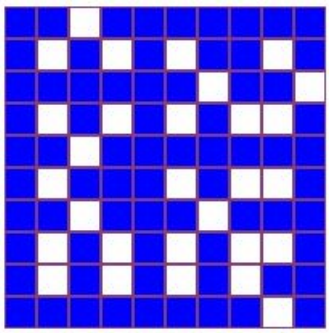

Figure 5.5: "Hierarchy" configuration with size 100.

- **Holes:** Consists of configurations that contain more bounded holes

than hierarchies reducing the interconnection between holes. See Figure 5.6 for an example.

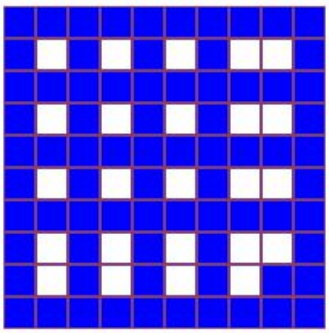

Figure 5.6: "Holes" configuration with size 100.

- **Medium:** Consists of configurations that share both hierarchies and bounded holes in a balanced way. See Figure 5.7 for an example.

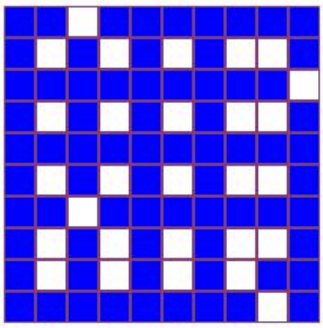

Figure 5.7: "Medium" configuration with size 100.

- **Minimal:** Consists of configurations that contain near to the minimum amount of modules necessary to maintain a bounding box of size n. See Figure 5.8 for an example.

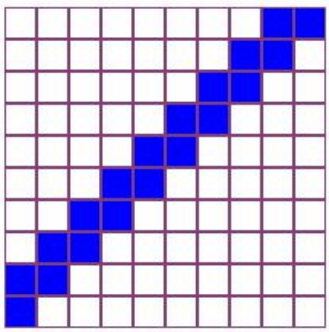

Figure 5.8: "Minimal" configuration with size 100.

- **Nested:** Consists of configurations whose graph is a tree and that nest all the leaves within the interior holes of the configuration, obliging the moving modules to connect critical pairs. See Figure 5.9 for an example.

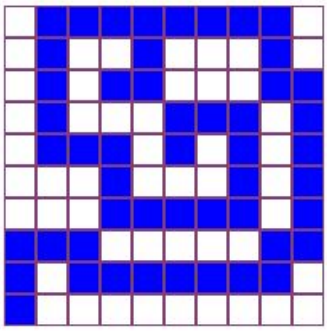

Figure 5.9: "Nested" configuration with size 100.

Tests have been developed for each type of configuration with different sizes. To see the evolution in computation time, the following sizes of the grid have been used: 100 cells, 1.024 cells, 3.025 cells, 6.400 cells and 10.000 cells. For the number of steps has been computed using configurations with the same amount of modules being the amounts: 100, 1000, 2250, 3500, 5000. The results as for computation time and number of steps are presented in figures 5.10 and 5.11 respectively.

Type \ #Cells	100	1024	3025	6400	10000
Dense	0.063	0.657	5.529	34.992	60.628
Hierarchy	0.005	0.411	5.585	22.612	60.249
Holes	0.003	0.485	5.585	34.257	62.222
Medium	0.004	0.519	5.585	22.214	59.853
Minimal	0.002	0.046	0.216	0.711	1.637
Nested	0.004	0.22	3.457	10.868	25.723

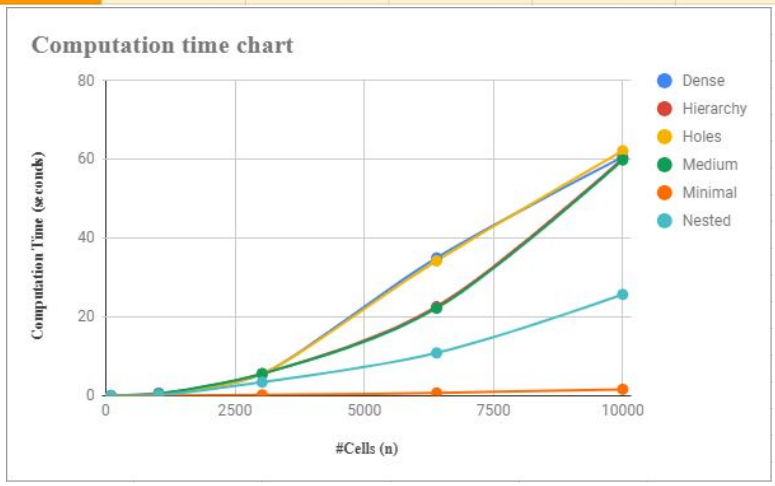

Figure 5.10: Mean computation time results for each input test.

Type \ #Mods	100	1000	2250	3500	5000
Dense	956	12174	151182	127047	554868
Hierarchy	1428	17045	163135	138651	353215
Holes	1111	17241	118477	131995	579443
Medium	808	27491	117543	118319	554274
Minim	7806	621082	2877678	6760828	13156460
Nested	3061	122339	505382	1034488	2843997

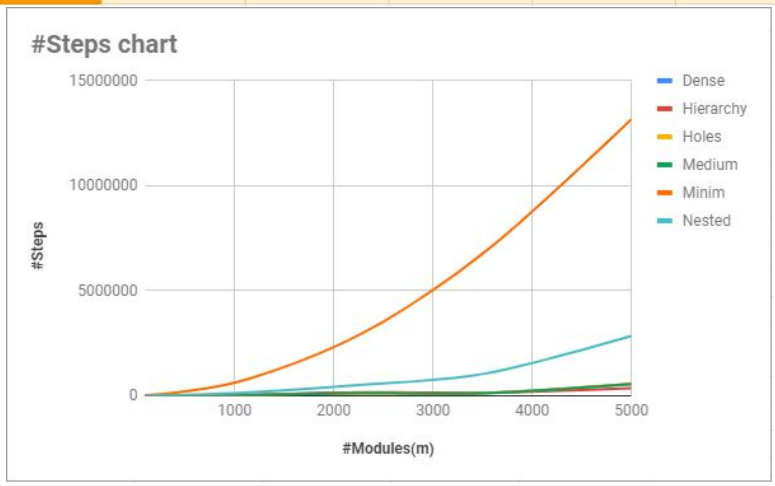

Figure 5.11: Number of slide moves in each input test's reconfiguration.

Seeing the results for each case and the evolution along the increment in size some conclusions can be extracted. The first conclusion extracted from the results is that our algorithm can effectively manage an input grid of size 10.000 since its computing time is 60 seconds. Secondly, as expected, using a minimal number of modules a grid size requires has both the lowest computation time and number of steps. Comparing this results with the results of the rest of tests we can see that the number of modules in the grid may affect more than the size of the grid does.

The third thing we can observe is that in both time and number of steps, the types: Dense, Hierarchy, Holes, and Medium, are not as different as we expected to see. However this could be expected since the number of modules in each of this types can be quite similar. Having more bounded holes than pseudo-holes, or a balanced mix of both does not make a significant difference. Another observation is that the great amount of modules in the dense configuration implies that a great amount of the modules might already be in-place. This reduces the amount of modules we need to move and the amount of cells we need to fill, thus reducing both execution time and number of steps.

Despite the number of holes or size of hierarchies not affecting the reconfiguration drastically, the way the holes are disposed does. In order to enclose all the movable modules with other modules, the configuration must not be dense. This reduces the number of leaves in the configuration making it easier to control them. Another important thing is that by nesting the movable modules inside a deep pseudo-holes' hierarchy, we oblige the modules to reduce such hierarchy in order to liberate modules in the outer boundary. This causes some extra moves of modules from the inside to the outside of the configuration that may vary depending on how deep the hierarchy is. Aside from that, since it is not a dense configuration, modules have to slide over a great amount of other modules in order to get to the desired cell.

And last but not least, since the minimal configurations require the modules to slide over all the others, we can see how this affects drastically to the number of moves the robot does during the reconfiguration.

Chapter 6

Methodology

Once we have described the project and its objectives, what remains are the methods that will be used. The main part of the project consists on implementing an algorithm that solves the problem. For this same purpose the first task that we undertook was the study of the environment we would be working with, specific features and achievable goals. It was after this research that we decided the parameters of the problem on the thought of the short time in which the project will be taking place, but, choosing values that could be changed further on in the project for a more extensive and profound work on the topic. This allowed us to state a problem solvable within the given period of time while leaving open doors to its expansion.

Now that the problem is defined the first task on the list is to design a functional algorithm that is able to bring the robot system from one configuration to the other without taking much on account the number of steps made or the computing time. And once we achieve a valid algorithm that solves the problem we will be able to work on its different optimizations. This allows us to focus first on getting to the core of the project ensuring the reach of our goals before the deadline, and then giving the time needed for a more profound study. After the main objectives of the project are reached, the future steps will most likely be changing the shape of the modules, and in case of time being left, work on the design of a distributed algorithm.

For a further detailed schedule, we found that the algorithm can be divided in three different sections: pre-processing, filling the bottom-most row, filling the rest of the rows. This distribution has been reached after analyzing the process that the algorithm should do. Of these three parts, we will start working on the second one since it's the base of the algorithm. The filling of the first row includes some issues that the filling of the rest of the structure does not in case the first row is already built, and after

attaining all the information that we need to undertake the process of filling we will be able to design a proper pre-processing.

For all of that, every two weeks a meeting with the director will be agreed in order to keep an eye on the development of the project and to give further advise on the next steps or corrections on the work done. In the meetings not only the advances but the proof and demonstrations of the validity of them will also be explained.

6.1 Development tools

For the development of this project there has been no need for external programs on the matter of algorithm implementation, the code has been done in *JavaScript* since it is a quick coding language for easy prototyping and it is the main core of the web visualizer besides the HTML document. Additionally, for the web page we used a *JavaScript* library called *P5.js* in order to visualize the modules' movements on screen.

The main project manager that we have used is *Git* which has allowed us to track all changes, fixes and extensions of the code so that at the end of the project we have been able to evaluate the expected scheduling with the real one.

Last but not least, the writing of all the deliverables and reports have taken place in *Overleaf*, a *LaTeX* online editor, in order to give them the proper structure and image they require.

6.2 Obstacles and problems

When planning this project we considered possible obstacles and problems that could appear, and distinguished a few sectors that could be affected by those.

On one side, during the design of the algorithm we thought possible the discovery of some features or characteristics of our specific problem that we had not taken into account. As long as these could have been profitable features, they could have also been a problem obliging us to edit some already tested parts in order to use these features for improvement or to avoid them. However, no such problems have arised due to the prior systematic study of the problem and its possible solutions.

After designing the algorithm, its implementation started by the development of its 2nd part, the filling of the 1st row, as it was planned. However,

that changed after realizing which data had to be taken into account about the structure of the configuration. In order to establish then a first data structure model, we decided to move to the implementation of the 1st part, the preprocessing, allowing us to know which data could be extracted from the input and how.

Another part that could have been affected was the testing. The design of good tests that ensured the reliability of the algorithm and its well functioning had to be done with accuracy. In the end the best way of creating input tests was to start with the basics and evolve them by adding the new special cases found along the testing. Moreover, using this method, we ensured that by solving a new problem, we did not break any of the previous solutions.

Further in the matter of testing, we found ourselves with the online visualizer that we also implemented. That was at first an obstacle in the matter of time. Due to the little expertise in the topic, we did not know how much would it take. Despite this, we found a suitable library for *JavaScript*, *P5.js*, which helped develop the web page in less time than the expected. With this found component we were able to treat the input from users and output of the reconfiguration algorithm for it to be displayed.

6.3 Project Scheduling

The project started on February 11th and ends on June 27th. The plan has been to devote 4 months of work to this project as planned. This time restriction made it important to plan all tasks to be done during the project and schedule them properly. For that we defined all the tasks to be done and established their deadlines.

6.3.1 Tasks description

This section is devoted to the enumeration of the different tasks necessary for the project and a brief description of each one.

Information research

The first step consisted on the research of scientific articles, books and any documents about the topic. This step took place from the beginning of the course until the end of the first phase in the Project Management (GEP) course. Though the main part of this task was made during the described period. However, during the report writing, extra articles and information were searched for references or examples.

Contextualization and planning

This task corresponds with the GEP course and groups all the deliverable tasks that were made for it. These tasks consisted on looking for information that helped us situate the project within its field, in this case robotics and more precisely modular robots. After that we designed a plan for the project. First distributing the different tasks in time. Then we estimated a starting budget and took a few decisions towards the sustainability of the project.

Design of the algorithm

The project's main goal is to design an algorithm to reconfigure lattice-based modular robots in-place. It is for that reason that first of all we studied the possible solutions to the problem. In this study we were able to distinguish three parts in which our algorithm was divided: the pre-processing, the filling of the first row, and the filling of the rest of rows. At first we decided to change the order of these parts for design and implementation purposes. The plan was to begin with the design and implementation of the filling of the first row. But, for various problems on what data to use and how to store it we decided to follow the algorithm's order and start with the preprocessing.

Tests and fixes

Due to the short time we had and the complexity of the algorithm, the best way to work in it was the Waterfall methodology. The algorithm has been implemented one step at a time and tested at the end of the development of each part. This allowed us to make the most out of the time we had to work. This also reduced the time needed when putting all the parts together. In this task several and specific input test were designed for its use in experimentation with the algorithm.

Web page development

As a final task, after the algorithm is complete, for further testing of the algorithm we designed a web page. This allows us to input tests and see all the steps that the robot makes during the reconfiguration. This web page allows us to visually prove the effectiveness of our algorithm.

Writing the report

The task consisted in documenting the whole project explaining all the processes, decisions made, justifying all the work and finally describing the results and giving a conclusion to the project. In the report we also explain the conduct of the algorithm and how the whole program will work step by step.

6.3.2 Time partitioning

In this brief section compare the initially estimated time partitioning and the final time used for each task.

Task	Estimated time (h)	Dedicated time(h)
Information research	40	40
Contextualization and planning	75	75
Pre-processing	50	70
Filling of the first row	80	60
Filling of the rest of the rows	60	80
Tests design and fixes	50	50
Web page development	50	30
Writing the report	45	45
Total	450	450

Table 6.1: Table containing the estimated and dedicated time in hours for each task.

6.3.3 Needed resources

In order to make progresses in the project we needed some resources that we used to complete all the described tasks. These resources can be classified into three different classes.

Human resources

These resources refer to all the people involved in the project.

- **Project manager**: is the person who defined the project's plan, ensured its execution and wrote the main part of the report.
- **Algorithm designer**: is the person who designed and implemented the whole algorithm and the web page used for visualization.
- **Project director**: is the who supervised both the manager and the designer and advised them when problematic circumstances or doubts arose on specific topics.

Software resources

All the software programs and interfaces that has been used during the project. The software needed changed a little bit during the project. In the end *Lubuntu 16.04* was not needed since everything was done on windows,

and instead of an HTML editor we used a *JavaScript* library called *P5.js*, that served the same purposes.

- **Windows 10**: has been used in all tasks.
- **LaTeX**: used in the report writing process of all the reports made along the project.
- **Gantter**: tool used in the project planning for the tasks' schedule.
- **Visual Studio Code**: text editor for the code writing of both the algorithm and the web page.
- **GitHub**: project management software that has been used during the design and development period.
- **P5.js**: *JavaScript* library used for the algorithm visualization in the web page.

Hardware resources

All the hardware components that has been used to complete the tasks.

- **Personal Laptop**: (Intel Core i7, 8GB RAM, 256GB SSD) used within the whole project.
- **Web Server**: maintains the web page online for us to be able to demonstrate the algorithm's functionality.

6.3.4 Gantt chart

Since some changes have been made on the final scheduling we will now show the comparison between the designed schedule and the final one (see Figure 6.1).

	Activity	Predecessor	Start	End	Days
1	Information research		11-02-19	18-02-19	6.0
2	Contextualization and planning		19-02-19	25-03-19	25.0
2.1	Scope, Context, State of the Art		19-02-19	26-02-19	6.0
2.2	Project planning	2.1	27-02-19	04-03-19	4.0
2.3	Economic management and sustainability	2.2	05-03-19	11-03-19	5.0
2.4	Project initial presentation	2.3	12-03-19	25-03-19	10.0
3	Algorithm design and development		14-03-19	10-05-19	42.0
3.1	Preprocessing		14-03-19	02-04-19	14.0
3.2	Filling of the first row	3.1	03-04-19	18-04-19	12.0
3.3	Filling of the rest of rows	3.2	19-04-19	10-05-19	16.0
4	Tests design and fixes	3	13-05-19	24-05-19	10.0
5	Web page development	4	27-05-19	03-06-19	6.0
6	Report writing		04-06-19	27-06-19	18.0

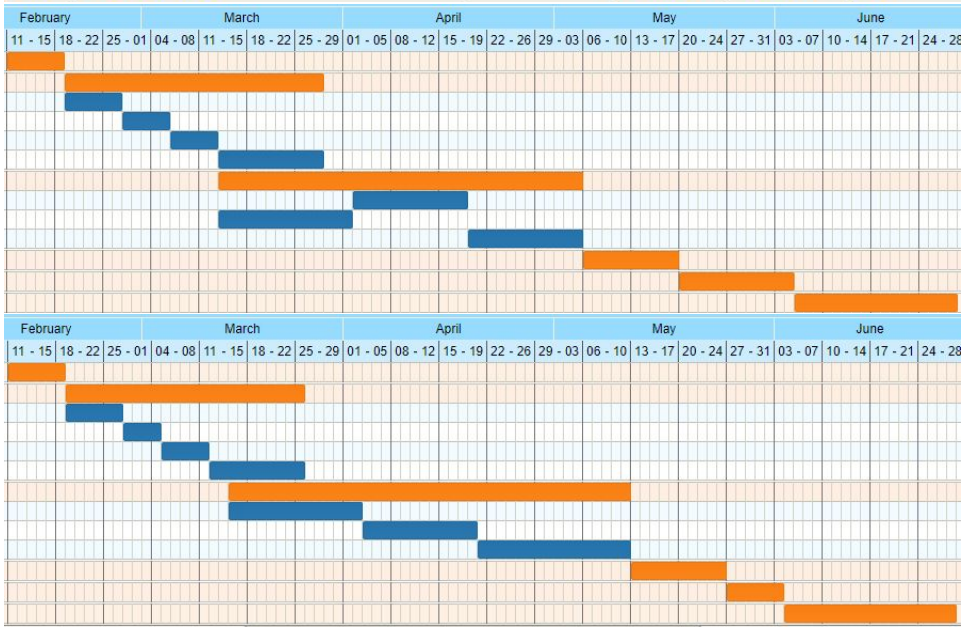

Figure 6.1: Information on each task and its subtasks followed by the estimated planning and the real one.

6.3.5 Action plan

The starting action plan in order to complete the project was to work only on one task at a time. This was followed during the project planing and the design, implementation and testing of the algorithm. However the creation of the web page and the report writing were undertaken at the same time in order to make the most of the remaining time.

As showed in the project final time distribution, during the algorithms design extra time had to be taken from tasks such as the filling of the first row or the web page development. This time was dedicated instead to tasks that required extra working hours like the preprocessing or the filling of the rest of rows. Despite this, the total amount of time used for the project did not change and no expansions were added to the project.

In addition to the tasks plan, every two weeks a meeting with the director of the project has been made in order to discuss the progress of it and point out and correct any issues that appeared along the processes. In some cases, when the deadline was getting close we decided to have a weekly meeting to have a more constant supervision of the project.

Despite the encountered problems on the algorithm design or time limitations, the paradigm of the original problem did not have to be modified. However, no further extensions of the project have been applied. These will remain as future steps.

To conclude this section, the amount of time dedicated to the project was 450 hours that we distributed in packs of 25 hours per week during 19 weeks. Thanks to the first planning of the project and the corrections and changes made along the process, we were able to complete the project and achieve the proposed goal.

6.4 Budget and economical management

6.4.1 Project budget

In this section we compare the budget estimated at the beginning of the project with the final expenses. To make the comparison clearer we compare separately each expenses section: hardware, software and human resources. To obtain the total budget used we also calculated the indirect costs in the project, such as electricity, internet and other needed materials, and the expenses of unexpected issues.

Hardware budget

In this project we used a few hardware components all through the different stages. During the project there have been no changes on the hardware used neither in which components nor how much did we use them. In the Table 6.2 we show the used hardware and how is the cost distributed.

Products	Price	Units	Useful life	Amortization
PC (OS included)	800 €	1	4 years	67 €
Total	800 €			67 €

Table 6.2: Hardware expenses distribution.

Software budget

Naturally, we also used a set of software programs to help us advance in our project. The following table displays the distribution of its cost.

Products	Price	Units	Useful life	Amortization
LaTeX	0 €	1	-	0 €
Gantter	0 €	1	-	0 €
VS Code	0 €	1	-	0 €
P5.js	0 €	1	-	0 €
GitHub	0 €	1	-	0 €
Total	0 €			0 €

Table 6.3: Software expenses distribution.

Human resources budget

The human resources prices were taken from [6] in order to estimate the needed budget. The changes made on the time distribution altered the hour distribution of the different roles in each task. For that in Table 5.5 and Table 5.6 we compare the total amount of hours and show the new time distribution. However notice that since most of the hours exchanged belong exclusively to the developer, the total difference is little.

For a better understanding of the previous results, Table 6.5 shows the time distribution of the different agents in each task.

Role	€/hour	Hours	Estimation	Final cost
Project Manager	37 €	90	3.330 €	3.330 €
Project Director	35 €	60	2.030 €	2.100 €
Software Developer	26 €	300	7.852 €	7.800 €
Total			13.212€	13.230 €

Table 6.4: Human resources distribution.

Task	Duration	Roles		
		Manager	Director	Developer
Information research	40 h	20 h	5 h	15 h
Context and plan	75 h	45 h	10 h	20 h
Pre-processing	70 h	0 h	10 h	60 h
First row filling	60 h	0 h	5 h	55 h
General filling	80 h	0 h	10 h	70 h
Tests and fixes	50 h	0 h	5 h	45 h
Web page development	30 h	0 h	5 h	25 h
Writing the report	45 h	25 h	10 h	10 h
Estimated	450 h	90 h	58 h	302 h
Total	450 h	90 h	60 h	300 h

Table 6.5: Working time of each role in all tasks.

Indirect costs

On the working process a lot of costs that are not usually noticed exist: electricity, internet, office supplies, etc. We took all of these into account when estimating the total budget and have been monitorized along it. When doing the estimation, due to the price being different among the different companies and countries, we took the prices from the referenced companies of Spain [10, 11, 12]. The changes in the time schedule did change the working time of each role in each task, but, the use of our laptop, internet or other services did stay the same. For that no changes were made to to the indirect costs.

Unexpected costs

We computed an extra budget for unexpected expenses. However, the problems surged among time distribution only amount to the difference between the estimated cost and the final cost. This adds up to 18€ of extra expenses which can be completely covered by the unexpected costs budget created. That also means that we spent 1.437€ less than expected. See the expectancy and the total budget used in the Table 5.7.

Resources	Price	Units	Total cost
Electricity	0,119893 €/kWh	600 kWh	72 €
Fiber Optic 100MB	15 €/month	4 month	60 €
Web Page Server	4 €/month	1 month	4 €
Office supplies	50 €	-	50 €
Total			186 €

Table 6.6: Indirect expenses.

	Total cost
Estimated	1.455 €
Used	18 €
Rest	1.437 €

Table 6.7: Contingency expenses and rest.

Total budget

Once explained the budget distribution, the next table shows the comparison between the estimated and the final budget of this project.

Resources	Estimated	Final
Hardware Resources	67 €	67 €
Software Resources	0 €	0 €
Human Resources	13.212 €	13.212 €
Indirect Costs	186 €	186 €
Contingency	1.455 €	18 €
Total	14.920 €	13.483 €

Table 6.8: Total expenses.

As we can see, due to the previous project planning, the final expenses were less than the estimated ones. The problems encountered were coped with in a way that they did not affect much to the project expenses. Therefore, only 18 € from the contingency budget were used and the total cost was reduced by 1.437€ .

6.4.2 Budget monitoring

When planning the project, we scheduled the tasks, estimated an initial budget and foresaw possible problems that could occur during the project.

As we had foreseen no extra expenses were caused by hardware malfunctioning. On the software topic we used extra software than the one described but it acquired for free. Finally the foreseen expenses in problems on human resources were were larger than the real ones. For this, the budget did not need to be extended.

To conclude, as we planned at the beginning of the project, we now calculate the deviation of the cost using the following equations:

$$Cost\ deviation = EC - RC\ Consumption\ deviation = EH - CH$$

The previous parameters referred to Estimated Cost(EC), Real Cost(RC), Consumed Hours(CH) and Estimated Hours(EH).

$$EC = 14.920 \quad RC = 13.483 \quad CH = 450 \quad EH = 450$$
$$Cost\ deviation = 14.920 - 13.483 = 1.437€$$
$$Consumption\ deviation = 450 - 450 = 0\ hours$$

This means that by keeping the working hours the same as the estimated time, we reduced the the costs by 1.437€ . The cost deviation is not small, however since it is a positive value it does not affect the project in a negative way. Despite all of that it is possible that with a further foresight on the problems that could have arisen a closer to reality estimation could have been done.

6.5 Sustainability

In order to evaluate the sustainability of the project we decided to analyze its repercussion in the different fields: environmental, economic and social.

6.5.1 Environmental sustainability

To be able to make any progress in our project we needed to use a PC which needed to be plugged and consumed electricity. Indirectly to work in proper conditions we made use of light during the whole research, writing, implementing, etc.

In order to affect the environment as little as possible we took some measures towards electricity consumption. To keep the electricity consumption to its minimum we took care of the office lights and used them only when needed not to force our sight. The computer has been strictly used for work purposes so its electricity consumption was held to its minimum. Last but not least no other electronic devices that needed to be plugged in the office were used.

As for other factors involved, only one computer has been used and shared among the different human agents involved in the project. Aside from that, we only printed the documents that had to be delivered physically. Any document sharing has been done online and the use of paper has been held to its minimum. As for electronic residues, the laptop will be reused for future projects.

When approaching the useful life of the project, since the product developed is an algorithm, in the future it can be updated or modified. That would cost on extra consumption of the same tools used for this project. However, further updates or the maintenance of the project along time will not cost as much as the development of the whole project and will not have a specific impact on the environment.

To end, about the risks, it is possible that due major changes on the software used for the project, we need to update the algorithm's code in order to maintain its functionality. Despite this, the maintenance of the project should not be a major risk neither for the project itself nor the impact on the environment.

6.5.2 Economic sustainability

During the budget planning of this project, research on the price of the different resources had been done. In all cases the most fitting option with the least economic impacts had been chosen both in products, services, and human resources. The project was scheduled thoroughly so it was possible to complete it within the time given. As explained in the previous sections some time distribution problems appeared but due to a good first planning, solving them kept the costs within the contingency budget.

The products chosen have been held to only the needed ones. Since we reproduced the tests in simulations instead of the physical robots, the costs were the minimum ones. Aside from the products, their consumption has been held to a minimum and both the electrical and optic fiber are the least necessary for the project to move on. Last, the web page server expenses has been taken on the minimum unit for its exclusive use at the end of the project during a month.

6.5.3 Social sustainability

On a personal point of view, this project has allowed me to experience the whole development of a project going through all planning and implementing stages. This is a great aspect to introduce myself on the way I will have to approach my future researches and projects.

On the other hand, the branch of robotics studied in this project is a hot topic nowadays. Many researchers and engineers working in robotics could be interested on the results of this project and any aspects discovered during the whole process. Moreover, the project being the design of an in-place algorithm gives it further need to be developed since during the research not many in-place algorithms were found.

Last but not least, the development of this project does not affect our society in a direct way. Any results in this project would only affect the knowledge in this field, maybe giving ideas to improve robot systems that already exist or encouraging further investigation on the topic. But in any case none of the discoveries or results in this project could be harmful in this aspect.

6.5.4 Sustainability Matrix

After the analysis of the project and its different aspects, the resulting sustainability matrix is the one showed below. The overall sustainability score of this project is 64 out of 90.

	PPP	**Useful life**	**Risks**
Environmental	Design consumption	Ecological footprint	Environmental risks
	9/10	17/20	-4/-20
Economical	Bill	Viability plan	Economical risks
	7/10	15/20	-2/-20
Social	Personal impact	Social impact	Social risks
	9/10	13/20	0/-20
Sustainability range	25/30	45/60	-6/-60
	64/90		

Table 6.9: Sustainability matrix.

Conclusions

In this project, we have designed, developed and implemented a reconfiguration algorithm for square-shaped lattice modular robots that move by sliding. Taking the algorithm PSA from [1] as a base we designed and implemented an algorithm with a different goal. This goal is to turn any configuration C into the configuration R_c resulting from filling from bottom to top and left to right the bounding box of C with the same amount of modules.

During the development, we encountered that the main issues appeared when no modules could reach the next cell c to fill. In such cases, before filling c we have to free modules which are able to reach c. After studying such specific cases we determined that the same way it was done in PSA with a few modifications would solve our problem. The reason why modifications needed to be done is that while in PSA all cells to fill are on the exterior of the configuration, we might find cells to fill that are inside our configuration.

After the design and implementation of the algorithm, in order to have visual proof of the algorithm correctness we designed a simulator online that allows to input any configuration and see which are the resulting steps of its reconfiguration. This simulator helped us to undertake practical experiments on our algorithm.

In addition to the implementation of the algorithm, we decided to study its cost and how the input size and structure affects the execution time and the number of moves of the modules. On the one hand, the cost was determined in a theoretical way breaking the algorithm into pieces and adding up the final cost of each piece. Finally the conclusion is that, if n is the number of cells of the input grid, and m is the number of modules in the configuration, the total computation cost is $O(n)$ and the number of slide moves of the modules of the robot is $O(m^2)$, which is optimal.

On the other hand, for the experiments we decided to distinguish six types of configuration structure that could affect the results of the algorithm. After running the algorithm with the different input tests and with

different sizes, we determined that the number of modules in the grid, can affect the computation time of our algorithm more than the size of the grid itself. However, another important factor found is the position of the movable modules at the starting configuration. By nesting the movable modules of a configuration in the depths of its pseudo-holes' hierarchies, we oblige them to find a path from inside to outside the configuration before they can reach the next cell to fill. The extra moves given by this together with the large paths the modules have to traverse when moving towards the goal cell makes this nested type of configurations perform the higher amount of moves compared to the others.

Having solved the problem exposed in this project, many future expansions are left open. One of them is to extend the algorithm to hexagonal-shaped modules. The hexagonal grid gives a completely different look to the map and lends the chance to develop a general purpose algorithm to melt both square and hexagonal modules' configurations. Another possible future project is to parallelize the movement of the modules, thus reducing the reconfiguration time. Finally, on the same line of movement parallelization, designing a distributed algorithm instead of a centralized one is also a future step to have in mind. And for further complexity, it is also possible to think about how to extend the results to the 3-dimensional setting.

Aside from all the sections involved with the design and development of the algorithm, in this project there has also been a rigorous planning and study of the projects effects in different fields. During the planning of the project many things had to be managed such as time or budget. The time scheduling helped during the project to know which was the goal, how close we were to achieve it and overcome any problems arisen such as the delay of a task's end. Furthermore, the proposed methods towards reducing the impacts on the economic, environmental and social aspects have been taken into account, contributing to the viability of the project.

As for personal experiences, this project has been a first approach towards what a technical project is meant to be. As an engineer with huge future projects involving computer engineering together with other branches of science, this has been a first step towards achieving such projects in the right way. In addition, in this project I had to learn a few components that were completely new to me what gave me further experience in the vast field that is computer science.

Glossary

- **Configuration**: Distribution of the modules on the grid.
- **Configuration matrix**: Matrix that contains certain information of all the cells in the grid.
- **Critical pair**: Set of two modules aligned diagonally without any adjacent module connecting them. Can be found in pseudo-holes that are not holes.
- **Edge**: Side of a module, which can be interpreted as an edge in the boundary.
- **Empty Space Graph**: Adjacency graph of the empty cells in the grid.
- **Fill**: Action of moving a module to a goal empty cell.
- **Grid**: Rectangular space divided into square-shaped cells delimited by the minimum containing box of the starting configuration.
- **Hole**: Any vertex connected component of empty space cells.
- **Hole boundary**: Chain set of modules' edges that are incident to a hole.
- **Modules Graph**: Adjacency graph of the modules of the configuration.
- **Profile**: Silhouette of a configuration.
- **Pseudo-hole**: Any edge connected component of empty space cells.
- **Space**: Empty cell in the grid.
- **Traversal boundary**: Chain of edge connected modules adjacent to a hole.
- **Unbounded space**: Unbounded connected component of the empty space graph.

Bibliography

[1] A. Dumitrescu, J. Pach *Pushing Squares Around*, Graphs and Combinatorics 22:37-50, 2006. Prelim. version in SoCG 2004.

[2] Z. Abel, S. D. Kominers *Universal Reconfiguration of (Hyper-)cubic Robots*, arXiv:0802.3414v3 [cs.CG], 2011.

[3] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, G. S. Chirikjian *Modular Self-Reconfigurable Robot Systems*, IEEE Robotics & Automation Magazine, 2007

[4] T. Larkworthy, S. Ramamoorthy *A characterization of the reconfiguration space of self-reconfiguring robotic systems*, Robotica, 29:73-85, 2011. Prel. version in ICRA 2010.

[5] H. Ahmadzadeh, E. Masehian *Modular robotic systems: Methods and algorithms for abstraction, planning, control, and synchronization*, Artificial Intelligence, 223:27–64,2015.

[6] Hays plc. *Un análisis de salarios y sectores en españa - Guía del mercado laboral 2018*, URL: https://www.esic.edu/empleabilidad/pdf/recursos/guia-hays-2018-sectores-y-salarios.pdf (visited on Mar. 11, 2019).

[7] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, G. S. Chirikjian *Modular Self-Reconfigurable Robot Systems*, IEEE Robotics & Automation Magazine, 2007

[8] F. Hurtado, E. Molina, S. Ramaswami, V. Sacristán *Distributed reconfiguration of 2D lattice-based modular robotic systems*, Auton Robot, 38:383-413, 2015.

[9] B. Kwon An, *EM-Cube: Cube-Shaped, Self-reconfigurable Robots Sliding on Structure Surfaces*, IEEE International Conference on Robotics and Automation, 2008.

[10] Movistar. *Fibra óptica 100MB - ADSL y Fibra óptica*, URL: https://www.movistar.es/particulares/internet/adsl-fibra-optica/ (visited on Mar. 10, 2019).

[11] Hostalia. *HOSTING WEB - Completos planes de hosting para alojar tu web*, URL: https://hostalia.com/hosting (visited on Mar. 10, 2019).

[12] Endesa Inc. *Tarifa one luz*, URL: https://www.endesaone.com/one-luz (visited on Mar. 10, 2019).